Toward Measures for Software Architectures

Gary Chastek
Robert Ferguson

March 2006

Software Engineering Measurement and Analysis

Technical Note
CMU/SEI-2006-TN-013

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

Abstract

This technical note describes the results of a preliminary investigation into measures for software architecture. It focuses on measures that directly indicate the health of or detect a problem with the software architecture of an up-and-running software system.

Defining these architectural measures is very difficult. The software architecture deeply affects the subsequent development and project management decisions, such as the breakdown of the coding tasks and the definition of the development increments. Most existing measures for up-and-running software systems capture the cumulative results of architectural, developmental, and managerial decisions and do not directly address the health of the software architecture.

The investigation into measures requires the joint participation of the software architecture and measurement communities. Since the software architecture community has made such rapid progress over the past ten years, this report first describes what the measurement community needs to know about software architecture to understand the difficulty of defining architectural measures. The current relevant literature is then described in terms of its potential contribution to this research. Finally, the report identifies areas for future research into the application of measurement technology to software architectures.

The ultimate goal of this body of work is to provide measurement guidance and quantitative decision support to software practitioners, including software architects and project managers.

1 Introduction

Organizations develop software to address their business and market goals. Such goals might include the ability to

- increase market share by developing new products faster
- improve price margin by developing new products at a reduced cost
- customize new products to meet a particular customer's needs

Our understanding of the importance of software architecture and its role in achieving such organizational goals has grown dramatically over the past ten years. The success of a software development effort is critically dependent on the effectiveness of the software architecture. Effective software architectures

- support communication among the system stakeholders by providing a common language for the project manager, coder, end-user, customer, and others
- document the early design decisions that critically and disproportionately affect all the subsequent development efforts
- provide an intellectually-manageable abstraction of a system that is transferable to other similar developments [Bass 03]

In this technical note, we identify measurement needs for both the system architect and the project manager. We focus our attention on development projects that extend an existing system, requiring the architect and project manager to evaluate the need for changes to the architecture against the cost of that work.

Several authors, including Daniel Paulish in his book *Architecture-Centric Software Project Management* [Paulish 02], have made a strong connection between the software architecture and the development of software products. For example, the software architecture determines the

- required skills for the developers (e.g., programming languages and software tools)
- breakdown of the coding effort (e.g., number of programmers required)
- development increments (e.g., what pieces of the software must be completed to produce an increment)

It is precisely these connections that make measuring software architectures difficult. Existing project management measures capture the cumulative effect of multiple factors, including the effectiveness of the

- software architecture and its documentation
- project management
- developers
- tools used
- process

However, the effect of software architecture decisions is so pervasive in the development that these project management measures cannot satisfactorily isolate problems with the architecture from other development problems.

For example, did a product change request arise from problems with requirements or problems with the architecture? Did the architecture documentation properly convey the design to the developers? Did the developers conform to the architecture? Did the developers properly implement the software? Is the development process flawed? The challenge of this work is to distinguish between required changes to the software architecture and other required changes.

Given the impact of the software architecture on a software development, it is reasonable to quantitatively monitor how well the architecture is achieving the organization's goals. That is, how can a software practitioner[1] conclude or prove that a software or development problem requires a change to the software architecture? Architectural measures are needed that directly indicate when a change is required in the software architecture, or that verify that the software architecture satisfies its goals.[2] This report identifies areas for research to achieve such measures.

1.1 Scope and Roadmap

This technical note describes our preliminary investigation into determining the appropriate measures to apply to the maintenance of a software architecture. It does not solve the problem of quantitatively identifying required changes in a system's software architecture, but it does identify areas of research that could result in the definition of such measures. The key issues to be addressed by further research are

- the difficulty in isolating the effects of the software architecture on a development
- the complexity of the organizational, business, and market context in which a software architecture exists

The ultimate goal of this work is to provide comprehensive measurement guidance for software practitioners as they develop, manage, and maintain their software architectures.

[1] "Software practitioner" is used to indicate the key software development roles, such as requirement engineer, architect, coder, tester, and project manager. This report focuses on the architect and project manager roles, but other software practitioners are equally relevant.

[2] This represents our current working definition of architectural measures.

Section 2 of this report describes the software architecture definitions and concepts used. Section 3 describes measures in the current literature and their relation to our notion of architectural measures. Section 4 presents our conclusions and describes potential areas for future work.

2 Software Architecture

Because contemporary software systems are large and complex, their development and maintenance require the efforts of many people with varying skills. So, who does what, and when? The software architecture, when properly conceived and documented, provides a technical plan that answers this question for the development of today's software systems.

This section provides, from a measurement point of view, a minimal and informal description of software architecture and how it is designed. It defines the terminology used in later sections and provides non-architects with some insight into the software architecture and its development.

It is neither a complete nor comprehensive introduction to software architecture. Such descriptions are available for project managers in *Architecture-Centric Software Project Management* [Paulish 02], and for architects in *Software Architecture in Practice* [Bass 03], and *Documenting Software Architectures: Views and Beyond* [Clements 02].

Section 2.1 defines software architecture and views, which are the models used to document the software architecture. Section 2.2 defines quality attributes, which are the key inputs to the architecture design process. Section 2.3 describes the architecture design process based on a particular design method, the Attribute-Driven Design Method (ADD) [Bachmann 00, Bass 01].

2.1 Software Architecture Concepts and Definitions

There are many different definitions of software architecture. In *Software Architecture in Practice,* Bass defines software architecture as "the structure or structures of a system, which comprise software elements, the externally visible properties of those elements, and the relationships among them" [Bass 03]. Examples of such elements could include compilation units and processes, each with its own related structure.

A software architecture is typically documented using multiple views. A "view" is described by Clements as "a representation of a set of system elements and the relationships associated with them" [Clements 02]. Together, these definitions are saying that the software architecture serves multiple purposes and hence cannot be captured in a single model (i.e., a view).

In his seminal paper “The 4+1 View Model of Architecture,” Kruchten proposed the use of the five following views:

- the **logical view** that supports the system’s services provided to the end-user
- the **process view** that describes the synchronization and concurrency aspects
- the **development view** that supports construction of the system and management of its development
- the **physical view** that maps the elements of the previous three views onto processing nodes
- a **fifth view** that ties the other views together by a set of scenarios describing how the elements of the other views cooperate [Kruchten 95]

Other sets of views have been proposed in such works as “Software Architecture in Industrial Applications” [Soni 95] and *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise* [Herzum 99]. Clements writes that even more views are possible and necessary [Clements 02].

Thus there are multiple abstractions (i.e., elements and their relationships) associated with a given software architecture. The abstractions that are useful for a specific architecture might not be useful for another. Since the potential representations of the architecture differ the basic measures will differ also. There is much work to reconcile these differences before we can create a common understanding for a useful set of architectural measures. In other words, it is difficult enough to determine “why” and “what” to measure. The multiple representations will further confound the difficulty of any implementation.

2.2 Quality Attributes

ISO/IEC 9126-1:2001, Software engineering - Product quality - Part 1: Quality model defines “quality” as “a set of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs” [ISO/IEC 98]. Once referred to as “nonfunctional requirements,” quality attributes are the realizations of a product or service’s quality. That is, quality attributes are the specific software system characteristics that combine to produce system quality. Examples include performance, modifiability, flexibility, security, and predictability. Chung provides an extensive listing and description of quality attributes in *Non-Functional Requirements in Software Engineering* [Chung 00].

Since the software architecture is the principal enabler of quality attributes [Bass 01], a great deal of software development research has been done on quality attributes. For example, the Attribute-Driven Design Method (ADD) is a systematic method for refining qualities in the context of the system being designed and matching those refined quality attributes to the appropriate functionality [Bachmann 00, Bass 01]. A similar technique has been developed for early requirements engineering for software product lines [Chastek 01].

Quality attributes arise in the business case, market analysis, and requirements, and can refer to the product being developed or to the development of that product [Chastek 03]. Such quality attributes tend to be global, referring to the entire product or the development of that product. For example, a business might have goals of producing products that are easy for customers to use and of producing new products quickly. The first goal refers directly to the product, while the second refers directly to the entire product development process—and to the software architecture, which must support such rapid development.

This again gives rise to a variation-related problem for architectural measures. For example, two systems might require adaptability as a driving quality attribute, but what "adaptability" means for each system can be quite different. In other words, the meanings of quality attributes are context-dependent. This variation complicates the definition of a generally applicable and useful set of architectural measures.

2.3 Design Process

This section, loosely based on ADD, describes architecture design as a decision-making process. Initially the architect identifies the driving requirements—that is, the quality attributes that are most critical to the success of the software architecture.

At each point in the design process, the following must be considered:

- the context of the previous design decisions
- the functionality to be provided
- the set of quality attributes relevant to that functionality

In order to make sensible design choices for the software architecture, the architect typically reasons using scenarios representing the product in context. For example, the design of a financial transaction-processing system could include a requirement that the response time for any transaction be no more than five seconds. That is, the elapsed time from when the end user enters a transaction until the system displays a result back to that end user must be less than five seconds.

The architect might select the "display account balance" transaction for his scenario. At this point in the design process,

- the context of the previous design decisions means that the "send the end-user request" and the "send the response to the end-user" parts of the transaction processing have been designed, and each of these requires a half second to complete
- the functionality to be provided is to "prepare the account balance report"
- the set of quality attributes relevant to that functionality is to "prepare the account balance for transmission to the end-user" in less than four seconds

Typically there will be multiple design techniques available to choose from, and the architect must now decide among them. These techniques can be captured in an architectural pattern

[Buschmann 96], or as a tactic [Bass 03]. Frequently the technique selected will involve a tradeoff between the quality attributes required by that functionality. For example, one technique may be more secure but slower, while another is less secure but quicker.

The design process transforms global system qualities into local realizations. For example, a quality attribute such as “rapid delivery of a new product to market,” which is defined globally in the business case, can be transformed into a related set of very specific quality attributes based on the current design context. The local realization can be to parameterize specific software components to reduce overall product development time. Ultimately, the architect must verify that the resultant architecture satisfies the required global qualities. The architect must think through the implications of those global qualities, transform them into intermediate and local goals, and make design decisions based on local goals to ensure that the software attains the global qualities.

2.4 Summary

The organizational context that affects the software architecture also affects the definition and use of architectural measures, including the

- business and market goals and constraints for the project
- domain of the project’s product(s) (e.g., telecommunications, financial)
- context for the organization (e.g., skill levels of project members, tools available)
- type of project (e.g., single-system, software product line)

From the measurement point of view, there is a good deal of variability surrounding software architecture. There are multiple models (i.e., views) used to document software architectures and multiple meanings for the drivers (i.e., qualities) of architectural design process. The models and the specific meanings of drivers used are both dependent on the specific organizational context, among other things. The tradeoffs and sensitivity points from the design process are also specific to a particular software architecture. This variability complicates the definition of widely-applicable architectural measures.

3 Architectural Measures in the Literature

The purpose of an architectural measure depends on the role of the person using it. This section explores the roles of the architect and the project manager as they relate to architectural measures and briefly describes some of the related work currently available in the literature.

3.1 Architectural Measures for the Architect

To the architect, an architectural measure is a diagnostic device used to

- determine the source of a problem in the software architecture. Specifically, what has to be changed in that architecture to address the identified problem?
- provide quantitative rather than qualitative design decision support
- monitor the design process

The paper "Using Service Utilization Metrics to Access the Structure of Product Line Architectures" is a good example of measures from the software architect's perspective [van der Hoek 03]. It defines two per-component measures to monitor the quality attributes optimality and variability: provided service utilization (PSU[X]) and required service utilization (RSU[X]). PSU(X) reflects the number of services provided by X and used by other components, normalized to the total number of services defined in component X. RSU(X) reflects the number of services required by component X but defined in other components, normalized to the total number of services required by component X.

The van der Hoek paper is significant because it defines its measures in terms of the architecture's abstractions and because it illustrates how existing modeling measures can be adapted to measure software architectures. For example,

- model-based measures such as those defined for the Rational Unified Process (RUP) can be adapted to the architectural abstractions relevant to the system being measured [Kruchten 01]
- object-oriented measures, such as those described in *A Data Model for Object-Oriented Design Metrics* [Abounader 97] and *Object-Oriented Metrics: Measures of Complexity* [Henderson-Sellers 95] can be used directly for objected-oriented architectural views or adapted for architectural views that are not object-oriented
- quality attribute measures, such as those described in "Metrics for Software Adaptability" [Subramanian 01], can be useful if properly tailored to the specific context

The book *The Engineering of Software Quality* [O'Brien 04] addresses the architect's need to monitor the design process by focusing on the application of measures to manage the incremental development of the software architecture. This book also touts the benefits of an architecture-centric development, from the developer's point of view. It discusses specific quality attributes, such as performance, reliability, scalability, and maintainability. The effect of each of these qualities on the design process is explained, including recommendations for measures that allow the architect to monitor the architecture relative to that quality.

3.2 Architectural Measures for the Project Manager

To the project manager, an architectural measure is a diagnostic or monitoring "device" to determine sources of problems in a software development that require a change in the software architecture. The project manager must

- make decisions about resource requests, tasking, and scheduling (which includes planning and change control)
- forecast completion dates and costs
- coordinate developer efforts
- ensure effective communication within the project
- monitor productivity and plan for the needed capacity and skills [PMI 04]

Since there are different stages of project management—planning, monitoring, and controlling—there are different corresponding measurement needs.

For **planning,** the project manager needs information about project size, how work will be partitioned among teams, and the development life cycle. The project manager might want to influence the decisions about these elements to balance specific risks inherent in the project, so a risk assessment of the technical plan is also needed.

For **monitoring,** the project manager needs warning indicators, progress indicators, and efficiency information.

For **controlling,** the project manager needs problem solving and planning information. Useful architectural measures in this area include size, complexity, coverage, completeness, and technical risk.

The book *Architecture-Centric Software Project Management* [Paulish 02] focuses on the role of a project manager in an architecture-centric project. The message of the book is that software architecture is intertwined with product development, and the project manager can either pay attention to the software architecture and reap the advantages or ignore the architecture and suffer the consequences.

Paulish defines several global measures, which are high-level indicators that span multiple phases of a project and indicate the overall condition of the project. For example, size, schedule deviation, productivity (measured as lines of code produced per day), defects, and customer change requests are defined in the context of an architecture-centric development.

However, little is said about measures that are directly linked to required changes in the software architecture.

3.3 Summary

Section 2 described the variability inherent in the design of the software architecture and how that variability argues for the use of measures that are tailored to the context of the specific organization. For example, a measure for the “flexibility” of a software system in the abstract is not as useful as one that measures precisely what the organization means by flexibility. Goal-driven software measurement [Park 96] is designed specifically to deal with the variation described earlier in this report. In any case, practitioners should use Goal-driven software measurement to validate any adapted architectural measure for use in their specific contexts.

4 Conclusions and Future Work

4.1 Conclusions

Software architecture is critical to the success of software development, yet little direct support for quantitative decision making exists in the literature. This report has examined some existing work that might be used directly or leveraged by the software practitioner.

- Goal-driven software measurement should be used for the deployment of architectural measures in a specific system.
- Existing modeling measures, such as those associated with RUP [Kruchten 01] and object-oriented modeling can be modified to measure software architectures.
- Quality-attribute-based measures found in the literature should be carefully applied to the measurement of software architecture.

Goal-driven software measurement should always be employed, either directly or to verify that a measure adapted from the literature is meaningful and useful in the specific organizational context.

There is a need for a comprehensive software practitioners' guide to architectural measures that would suggest appropriate measures to monitor the continuing effectiveness of a software architecture, during the system development and its evolution.

4.2 Future Work

A good deal of work still needs to be done to provide software practitioners with the measurement tools necessary for quantitative decision support. The following sections suggest a set of questions that must be addressed before effective guidance is possible—that is, before measure-based information can aid in the design process and continue to be helpful during system maintenance and evolution.

4.2.1 Improved Definition

Our working definition of architectural measures is, at best, preliminary. A better definition is needed that provides insight into how software practitioners can determine the appropriate measures to use in their specific contexts and for their specific software architectures.

4.2.2 Isolating the Effect of the Architecture

A key problem is determining when a development problem requires a change to the software architecture. A related area to investigate is conformance. Architectural conformance refers to how well an implementation adheres to the software architecture. If we knew a specific implementation conformed to the specified architecture, it would be easier to determine when a development problem is due to the architecture.

4.2.3 Architectural Design Support

Are there predictive architectural measures? Can past organizational information be leveraged to better design software architectures for new products? How can past architectural decisions be applied to future design decisions [Bachmann 02]?

4.2.4 Architectural Evaluation

What is the relationship between architectural measures and architectural evaluation? Are there quantitative measures that

- identify the need for an architectural evaluation?
- assign a confidence level to a particular application of an architectural evaluation?
- provide insight into the value of such an evaluation?
- monitor the results of an architectural evaluation, such as tradeoffs and sensitivity points, to ensure the continued health of the software architecture after such an evaluation?

4.2.5 Measures for Other Roles

How can architectural measures support decisions by executives? What are the ties between architectural measures, real options theory, and other economic modeling techniques? There is a wealth of published measures based on financial entities. Which of these measures could be used as is or adapted to architectural measures appropriate for use by an executive? How would an architect or project manager know when such information is of interest to the executive?

What are architectural measures for other developers, such as coders and testers? How can they recognize implementation problems that require a change to the architecture?

4.2.6 Translate Modeling Measures

An earlier section of this report discussed the adaptation of existing modeling measures for use with a specific software architecture. For example, complexity measures based on RUP [Kruchten 01] can be recast from the RUP abstractions to the architectural abstractions for a particular architecture. What existing modeling measures should be translated into

architectural measures? When would such translations constitute a valuable addition to the literature?

4.2.7 Adapting Goal-Driven Software Measurement

Can the context implied by the software architecture be leveraged to specialize Goal-driven software measurement for architectural measures? Can Goal-driven software measurement isolate the effects of a software architecture? What is the role of architectural scenarios in such a specialization? What are the indicators that a software architecture is healthy or faulty?

4.2.8 View-Based Measures

The Kruchten "+1" view ties the other views together through a set of scenarios describing how the elements of those other views cooperate [Kruchten 95]. Is there some special significance to those scenarios with respect to architectural measures?

4.2.9 Architectural Measures for System Evolution

Quality attributes are as important during system evolution as they are during development. New or different market demands or shifting business and organizational goals can alter the qualities that drive the software architecture. This in turn can trigger needed changes to that architecture. So, what measures could be devised to aid in the evolution of the software architecture?

4.2.10 Architectural Drift

Architectural drift is related to evolution and refers to the tendency, over time, for maintenance changes to the software architecture to cause the implemented architecture to deviate from its originally intended qualities. So, what measures could be devised to detect architectural drift and identify its cause?

References

URLs are valid as of the publication date of this document.

[Abounader 97] Abounader, J. & Lamb, D. *A Data Model for Object-Oriented Design Metrics*. Kingston, Ontario: Queen's University, 1997. http://www.cs.queensu.ca/TechReports/Reports/1997-409.pdf

[Bachmann 00] Bachmann, F.; Bass, L.; Chastek, G.; Donohoe, P.; & Peruzzi, F. *The Architecture Based Design Method* (CMU/SEI-2000-TR-001). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tr001.html.

[Bachmann 02] Bachmann, F.; Bass, L.; & Klein, M. *Illuminating the Fundamental Contributors to Software Architecture Quality* (CMU/SEI-2002-TR-025). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tr025.html.

[Bass 01] Bass, L.; Klein, M.; & Bachmann, F. "Quality Attribute Design Primitives and the Attribute Driven Design Method." *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*. Bilbao, Spain, October 3-5, 2001. London, UK: Springer-Verlag, 2001.

[Bass 03] Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice, 2nd Edition.* Boston, MA: Addison-Wesley, 2003.

[Buschmann 96] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; & Stal, M. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.* West Sussex, England: John Wiley & Sons, 1996.

[Chastek 01] Chastek, G.; Donohoe, P.; Kang, K.C.; & Thiel, S. *Product Line Analysis: A Practical Introduction* (CMU/SEI-2001-TR-001). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/publications/documents/01.reports/01tr001.html.

[Chastek 03] Chastek, G. & Donohoe, P. *Product Line Analysis for Practitioners* (CMU/SEI-2003-TR-008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tr008.html.

[Chung 00] Chung, L.; Nixon, B.A.; Yu, E.; & Mylopoulos, J. *Non-Functional Requirements in Software Engineering.* Boston, MA: Kluwer Academic Publishers, 2000.

[Clements 02] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2002.

[Henderson-Sellers 95] Henderson-Sellers, B. *Object-Oriented Metrics: Measures of Complexity*. Boston, MA: Prentice Hall, 1995.

[Herzum 00] Herzum, P. & Sims, O. *Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise.* New York, NY: Wiley, 2000.

[IEEE 00] IEEE Standard No. 1471-2000. *Recommended Practice for Architectural Description of Software-Intensive Systems.* IEEE Product Number SH9486, available for purchase at http://shop.ieee.org/ieeestore/.

[ISO/IEC 98] ISO/IEC FCD 9126-1.2. *Information Technology – Software Product Quality, Part 1: Quality Model*, 1998.

[Kruchten 95] Kruchten, P. "The 4+1 View Model of Architecture." *IEEE Software 12*, 6 (November 1995): 42-50.

[Kruchten 01] Kruchten, P. *The Rational Unified Process: An Introduction, 2nd Edition*. Boston, MA: Addison-Wesley, 2001.

[Losavio 04] Losavio, F.; Chirinos, L.; Matteo, A.; Levy, N.; & Ramdane-Cherif, A. "ISO Quality Standards for Measuring Architectures." *Journal of Systems and Software 72*, 2 (2004): 209-223.

[O'Brien 04] O'Brien, F. *The Engineering of Software Quality*. Sydney, Australia: Pearson Education, 2004.

[Park 96] Park, R.; Goethert, W.; & Florac, W. *Goal-Driven Software Measurement–A Guidebook* (CMU/SEI-1996-HB-002). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996. http://www.sei.cmu.edu/publications/documents/96.reports/96.hb.002.html.

[Paulish 02] Paulish, D. *Architecture-Centric Software Project Management.* Boston, MA: Addison-Wesley, 2002.

[PMI 04] Project Management Institute. *A Guide To The Project Management Body Of Knowledge*. Newtown Square, PA: November 2004.

[Soni 95] Soni, D.; Nord, R.; & Hofmeister, C. "Software Architecture in Industrial Applications," 196-207. *Proceedings of the 17th International Conference on Software Engineering*. Seattle, Washington, April 23-30, 1995. New York, NY: ACM Press, 1995.

[Subramanian 01] Subramanian, N. & Chung, L. "Metrics for Software Adaptability," 95-108. *Proceedings of the Software Quality Management Conference*, April 18-20, 2001, Loughborough, UK. http://www.utdallas.edu/~chung/ftp/sqm.pdf.

[van der Hoek 03] van der Hoek, A.; Dincel, E.; & Medvidovic, N. "Using Service Utilization Metrics to Assess the Structure of Product Line Architectures," 298. *Proceedings of the Ninth International Software Metrics Symposium*, Sydney, Australia, September, 2003. Los Alamitos, CA: IEEE Computer Society Press, 2003. http://doi.ieeecomputersociety.org/10.1109/METRIC.2003.1232476

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	March 2006	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Toward Measures for Software Architectures	F19628-00-C-0003

6. AUTHOR(S)

Gary Chastek, Robert Ferguson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2006-TN-013

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

This technical note describes the results of a preliminary investigation into measures for software architecture. It focuses on measures that directly indicate the health of or detect a problem with the software architecture of an up-and-running software system.

Defining these architectural measures is very difficult. The software architecture deeply affects the subsequent development and project management decisions, such as the breakdown of the coding tasks and the definition of the development increments. Most existing measures for up-and-running software systems capture the cumulative results of architectural, developmental, and managerial decisions and do not directly address the health of the software architecture.

The investigation into measures requires the joint participation of the software architecture and measurement communities. Since the software architecture community has made such rapid progress over the past ten years, this report first describes what the measurement community needs to know about software architecture to understand the difficulty of defining architectural measures. The current relevant literature is then described in terms of its potential contribution to this research. Finally, the report identifies areas for future research into the application of measurement technology to software architectures.

The ultimate goal of this body of work is to provide measurement guidance and quantitative decision support to software practitioners, including software architects and project managers.

14. SUBJECT TERMS	15. NUMBER OF PAGES
architectural measures	25

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL